# Package: simplaceUtil (via r-universe)

October 16, 2026

**Title** Provides Utility Functions and ShinyApps to work with the modeling framework 'SIMPLACE'

**Version** 0.9.1

**Date** 2026-05-19

**Description** Provides Utility Functions and ShinyApps to work with the modeling framework 'SIMPLACE'. It visualises components of a solution, runs simulations and displays results.

**License** GPL-3

**Encoding** UTF-8

**Depends** R (>= 4.1)

**Imports** utils, DiagrammeR, rlang, dplyr, shiny, stringr, tidyr, xml2, shinyFiles, DT, ggplot2, readr, simplace

**VignetteBuilder** knitr

**Suggests** knitr, rmarkdown, rsvg, DiagrammeRsvg

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**Config/pak/sysreqs** cmake libglpk-dev make default-jdk libicu-dev libuv1-dev libxml2-dev libx11-dev zlib1g-dev

**Repository** https://gk-crop.r-universe.dev

**Date/Publication** 2026-05-19 10:56:48 UTC

**RemoteUrl** https://github.com/gk-crop/simplaceUtil

**RemoteRef** HEAD

**RemoteSha** 18fdc9d7bc3fec0cf2972cab1ba7fda7366b1081

## Contents


addComponentInput . . . . . 3
addCSVOutput . . . . . 4
addMemoryOutput . . . . . 5
addOutputVariable . . . . . 5

addTimingSimComponent . . . 6
addUserVariable . . . 7
addXMLParameter . . . 8
changeAllOutputTypesToMemory . . . 8
changeInterfaceFile . . . 9
changeOutputType . . . 10
componentsToGraph . . . 10
createCodeStubsForSimVariables . . . 11
createResourceStubsFromCsv . . . 12
createResourceStubsFromXml . . . 13
enhanceFunctionWithBoundaries . . . 14
enhanceFunctionWithPenalty . . . 15
fetchDescriptionFromWebsite . . . 17
fetchSimComponentlistFromWebsite . . . 17
fetchSimVariablesFromWebsite . . . 18
filterEdges . . . 18
formatProjectFile . . . 19
formatSolutionFile . . . 19
getComponents . . . 20
getElementsFromSolutionFile . . . 21
getLinkingFromComponent . . . 21
getLinkingToComponent . . . 22
getLinks . . . 22
getMemoryOutputIds . . . 23
getNeighborhood . . . 23
getOutputFilenames . . . 24
getResourceVariables . . . 25
getSolutionFromFile . . . 25
getSolutionFromText . . . 26
getSolutionInfoAsDataframe . . . 26
getTextFromSolution . . . 27
getUserVariables . . . 27
getXMLParameter . . . 28
getXMLParameterIDs . . . 28
graphLegend . . . 29
parseDate . . . 29
plotLayeredOutput . . . 30
plotScalarOutput . . . 31
readXMLParameterFile . . . 31
removeComponentInput . . . 32
removeNonMemoryOutputs . . . 32
removeOutput . . . 33
removeOutputVariable . . . 33
replaceVariable . . . 34
replaceVariablesWithValues . . . 34
runSimplaceCustomGuiApp . . . 35
runSimplaceGuiApp . . . 37
scaleXMLParameter . . . 37

setInputValue . . . . . 38
setInputValueForCategory . . . . . 38
setXMLParameter . . . . . 39
simplaceUtil . . . . . 40
solutionToGraph . . . . . 40
swapComponents . . . . . 41
transformLayeredData . . . . . 42
unitsymbols . . . . . 42
writeSolutionToFile . . . . . 43
writeXMLParameterFile . . . . . 43

**Index** **44**

---

`addComponentInput` *Adds an input to a sim component.*

---

## Description

Notice: One can only add inputs that are defined by the sim component. If source is not given, then the parameter value is used.

## Usage

```
addComponentInput(
  sol,
  componentid,
  id,
  source = NULL,
  value = NULL,
  datatype = NULL,
  unit = NULL,
  description = NULL
)
```

## Arguments

| | |
|---|---|
| `sol` | solution object |
| `componentid` | id of sim component |
| `id` | id of variable |
| `source` | source of variable |
| `value` | value of variable (is used only if source is not given or NULL) |
| `datatype` | datatype (optional) |
| `unit` | unit (optional) |
| `description` | short description (optional) |

## Value

modified solution object

addCSVOutput *Adds a new CSV output to solution.*

## Description

Any output with the same id will be removed. The added output has no output variables. One has to add new variables via addOutputVariable.

## Usage

```
addCSVOutput(
  sol,
  filename,
  outputid,
  frequence = "DAILY",
  rule = NULL,
  resetrule = NULL,
  cachesize = 10,
  divider = ","
)
```

## Arguments

| | |
|---|---|
| sol | solution object |
| filename | name of the output file |
| outputid | id of new output |
| frequence | one of DAILY, YEARLY, BOOLEAN, COMPLEX |
| rule | optional rule, when frequence is BOOLEAN or COMPLEX |
| resetrule | optional resetrule, when frequence is BOOLEAN or COMPLEX |
| cachesize | optional cachesize |
| divider | character that is used as a divider for the csv file |

## Value

modified solution object

`addMemoryOutput` *Adds a new memory output to solution.*

## Description

Any output with the same id will be removed. The added output has no output variables. One has to add new variables via `addOutputVariable`.

## Usage

```
addMemoryOutput(
  sol,
  outputid,
  frequence = "DAILY",
  rule = NULL,
  resetrule = NULL,
  cachesize = 10
)
```

## Arguments

| | |
|---|---|
| `sol` | solution object |
| `outputid` | id of new output |
| `frequence` | one of DAILY, YEARLY, BOOLEAN, COMPLEX |
| `rule` | optional rule, when frequence is BOOLEAN or COMPLEX |
| `resetrule` | optional resetrule, when frequence is BOOLEAN or COMPLEX |
| `cachesize` | optional cachesize |

## Value

modified solution object

`addOutputVariable` *Adds output variable to an existing output*

## Description

Adds output variable to an existing output

## Usage

```
addOutputVariable(
  sol,
  outputid,
  id,
  rule,
  datatype,
  mode = NULL,
  unit = NULL,
  description = NULL,
  format = NULL
)
```

## Arguments

| | |
|---|---|
| sol | solution object |
| outputid | id of output where the variable should be added |
| id | name of the new variable |
| rule | rule for the variable |
| datatype | of the variable |
| mode | one of FIRST, LAST, AVG, SUM (optional) |
| unit | unit of the variable (optional) |
| description | short description (optional) |
| format | format string (optional) |

## Value

modified solution object

---

# addTimingSimComponent *Adds elements for timing the execution time of sim components (and transformers)*

---

## Description

Adds elements for timing the execution time of sim components (and transformers)

## Usage

```
addTimingSimComponent(
  sol,
  filename = NULL,
  componentlist = NULL,
  interfaceid = "automatic_timing_interface",
  outputid = "automatic_timing_output",
  simcomponentid = "AutomaticTiming"
)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| filename | optional filename to write the timing information to csv file |
| componentlist | optional list of component ids (if empty, all components will be timed) |
| interfaceid | id for the interface (optional) |
| outputid | id for the output (optional) |
| simcomponentid | id for the timing simcomponent (optional) |

---

## addUserVariable *Adds an user variable to the* variables *section*

---

### Description

Adds an user variable to the variables section

### Usage

```
addUserVariable(
  sol,
  id,
  value,
  datatype,
  unit = NULL,
  description = NULL,
  rule = NULL
)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| id | id of the variable |
| value | value of the variable |
| datatype | of the variable |
| unit | unit (optional) |
| description | short description (optional) |
| rule | optional rule |

### Value

modified solution object

addXMLParameter *Adds a parameter to a specific parameterset*

## Description

When the parameter already exists in the parameter set, it is removed and the new parameter is added at the end.

## Usage

```
addXMLParameter(paramdata, key, id, value = "", unit = "", description = NULL)
```

## Arguments

| | |
|---|---|
| paramdata | parameterdata (xml2 object) |
| key | named vector c(idname="value") to identify the dataset, the name is the id of the parameter and the value has to match the parameter's value |
| id | name of the parameter which should added |
| value | value to set (numeric or character vector) |
| unit | unit of the parameter |
| description | description of the parameter |

## Value

modified parameter set (xml2 object)

## Examples

```
params <- readXMLParameterFile(system.file("input", "crop.xml", package="simplaceUtil"))
getXMLParameter(params, c(CROPNAME="soy bean"), "ExtraTableYZZ")
params_new <- addXMLParameter(params, c(CROPNAME="soy bean"), "ExtraTableYZZ", c(1,2,4))
getXMLParameter(params_new, c(CROPNAME="soy bean"), "ExtraTableYZZ")
```

changeAllOutputTypesToMemory *Changes all non-MEMORY outputs from solution to MEMORY outputs.*

## Description

When running large amout of runs (e.g. calibration) it's recommended to avoid to write outputs on disk.

## Usage

```
changeAllOutputTypesToMemory(sol)
```

## Arguments

| | |
|---|---|
| sol | solution object |

## Value

modified solution object

---

`changeInterfaceFile` *Changes all non-MEMORY outputs from solution to MEMORY outputs.*

---

## Description

When running large amout of runs (e.g. calibration) it's recommended to avoid to write outputs on disk.

## Usage

```
changeInterfaceFile(sol, id, filename, keeppath = FALSE)
```

## Arguments

| | |
|---|---|
| sol | solution object |
| id | id of interface |
| filename | new filename |
| keeppath | if set to true, original path is kept and only filename is modified |

## Value

modified solution object

## changeOutputType *Changes the type of an output from CSV to MEMORY or vice versa*

### Description

Changes the type of an output from CSV to MEMORY or vice versa

### Usage

```
changeOutputType(sol, outputid, type, filename = NULL, divider = ",")
```

### Arguments

| | |
|---|---|
| sol | solution object |
| outputid | id of output to change |
| type | one of 'CSV' or 'MEMORY' |
| filename | name of the output file if type changes to 'CSV' |
| divider | character that is used as a divider for the csv file |

### Value

modified solution object

## componentsToGraph *Creates a graph from component and links dataframe*

### Description

Creates a graph from component and links dataframe

### Usage

```
componentsToGraph(
  comp,
  links,
  showinterfaces = FALSE,
 fillcolors = c(var = "white", interface = "#e0e0e0", resource = "#ffff99cc", alias =
   "#ffff99cc", transform = "#ffdd88cc", mgm = "#ffaa99cc", simple = "#ff9988cc", normal
     = "#ff7777cc", grouped = "#ff6655cc", output = "#aaaaaacc"),
  shapes = c(var = "oval", resource = "egg", simcomponent = "rectangle", output =
    "polygon", interface = "triangle"),
  ...
)
```

### Arguments

| | |
|---|---|
| comp | components dataframe |
| links | links dataframe |
| showinterfaces | if TRUE, include interfaces in graph |
| fillcolors | vector of colors, named by component's subtype |
| shapes | vector of shapes, named by component's type |
| ... | options passed to DiagrammeR::create_graph() |

### Value

graph object of class dgr_graph

---

## createCodeStubsForSimVariables

*Create java and xml code stubs from a csv file*

---

### Description

The function creates Java code stubs for defining the fields, adding variables and initialising values. It creates XML subs for parameter file, as well as resource definition for parameter and input data, inputs for simcomponent and outputs.

### Usage

```
createCodeStubsForSimVariables(
  variables,
  component = "MyComponent",
  outfolder = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| variables | dataframe or path to csv file with variables |
| component | name of SimComonent |
| outfolder | optional, if given code is written the folder's files |
| ... | parameters passed to read.table (e.g. sep, dec etc.) |

### Details

The dataframe / CSV file should have columns

- contenttype (one of: constant, input, state, rate, out)
- id
- description
- datatype (Simplace Datatype or corresponding Java Datatype)
- unit
- min
- max
- default

### Value

list of code snippets

---

`createResourceStubsFromCsv` *Creates XML stubs for interface and resource section from CSV file structure*

---

### Description

Creates XML stubs for interface and resource section from CSV file structure

### Usage

```
createResourceStubsFromCsv(
  filename,
  id,
  sep = ",",
  keyvals = NULL,
  arraycolumns = NULL,
  frequence = "DAILY",
  rule = NULL,
  data = NULL
)
```

### Arguments

| | |
|---|---|
| `filename` | name of the CSV file |
| `id` | id for the resource |
| `sep` | separator for CSV file |
| `keyvals` | named vector of column indices or names that act as key column |

| | |
|---|---|
| arraycolumns | vector of column indices or names that are arrays |
| frequence | frequence attribute for the resource |
| rule | rule attribute for the resource |
| data | optional data.frame (otherwise data will be loaded from file) |

## Value

list of two strings ('interface' and 'resource')

## Examples

```
stubs <- createResourceStubsFromCsv(
  filename = system.file("input","weather.csv", package="simplaceUtil"),
  id = "weather",
  sep =",",
  keyvals = c("CURRENT.DATE" = "Date")
)
cat(stubs$interface)
cat(stubs$resource)

stubs <- createResourceStubsFromCsv(
  filename = system.file("input","soil.csv", package="simplaceUtil"),
  id = "soil",
  sep =";",
  keyvals = c("vSoilType" = "soiltype"),
  arraycolumns = 6:20
)
cat(stubs$resource)
```

---

# createResourceStubsFromXml

*Creates XML stubs for interface and resource section from XML file structure*

---

## Description

Creates XML stubs for interface and resource section from XML file structure

## Usage

```
createResourceStubsFromXml(
  filename,
  id,
  keyvals = NULL,
  frequence = "DAILY",
  rule = NULL,
  xmlnode = NULL
)
```

### Arguments

| | |
|---|---|
| filename | name of the xml file |
| id | id for the resource |
| keyvals | named vector of column indices or names that act as key column |
| frequence | frequence attribute for the resource |
| rule | rule attribute for the resource |
| xmlnode | optional xml_node (otherwise xml will be read from filename) |

### Value

list of two strings ('interface' and 'resource')

### Examples

```
stubs <- createResourceStubsFromXml(
  filename = system.file("input","crop.xml",package="simplaceUtil"),
  id="soil",
  keyvals = c("vSoilType"=1)
)
cat(stubs$resource)
```

---

## enhanceFunctionWithBoundaries

*Modifies function to return a penalty value for parameters outside boundaries*

---

### Description

The method takes a function as well as values for lower and upper boundaries and returns a modified function. The modified function returns the value of the original function when the parameters are within boundaries and the penalty value otherwise.

### Usage

```
enhanceFunctionWithBoundaries(
  fun,
  l_bound,
  u_bound,
  penalty_value = Inf,
  boundary_fun = NULL,
  param_pos = 1,
  ...
)
```

### Arguments

| | |
|---|---|
| fun | function to be modified |
| l_bound | vector with lower boundary values |
| u_bound | vector with upper boundary values |
| penalty_value | value if parameter outside boundaries |
| boundary_fun | optional function for complex boundary conditions |
| param_pos | argument position of the parameter |
| ... | arguments passed to original function |

### Details

Optionally an own function can be supplied to calculate whether the parameter is valid. The boundary function must take 3 arguments: parameter, lower boundary and upper boundary and must return TRUE or FALSE.

A main use case of this method are optimisation / calibration tasks. If the optimisation method and the function to optimise are both ignorant to boundaries one can turn the function into a boundary sensitive one.

### Value

a modified function that considers boundaries

### Examples

```
sqrt_bd <- enhanceFunctionWithBoundaries(sqrt, 0, 10)
sqrt_bd(-1)
sqrt_bd(1)
sqrt_bd(11)
```

---

## enhanceFunctionWithPenalty

*Modifies function to return a penalised value for parameters outside boundaries*

---

### Description

The method takes a function as well as values for lower and upper boundaries and returns a modified function. The modified function returns the value of the original function when the parameters are within boundaries and a penalised function otherwise.

## Usage

```
enhanceFunctionWithPenalty(
  fun,
  l_bound,
  u_bound,
 penalty_fun = function(value, distance) (value + distance) * exp(1000 * distance),
  distance_fun = function(x, l, u) {
     max(0, (l - x)/(u - l), (x - u)/(u - l))
 },
  param_pos = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| fun | function to be modified |
| l_bound | vector with lower boundary values |
| u_bound | vector with upper boundary values |
| penalty_fun | function taking value and distance and returns value modified value depending on distance |
| distance_fun | function that takes x, l_bound and u_bound and computes the distance of x to the boundaries |
| param_pos | argument position of the parameter |
| ... | arguments passed to original function |

## Details

Optionally an own function can be supplied to calculate the penalised value. The function must take two arguments: the original value and the distance of the parameter to the boundaries.

Additionally an own distance function can be supplied, that has to take three arguments: the parameter, the lower boundaries and the upper boundaries.

A main use case of this method are optimisation / calibration tasks. If the optimisation method and the function to optimise are both ignorant to boundaries one can turn the function into a boundary sensitive one.

## Value

a modified function that changes value when parameter outside bounds

## Examples

```
sqr_bd <- enhanceFunctionWithPenalty(\(x) x^2, .1, 10)
sqr_bd(-1)
sqr_bd(1)
sqr_bd(11)
```

---

`fetchDescriptionFromWebsite`

*Fetches the Description (as HTML source code) of a SimComponent from Simplace Website*

---

## Description

Fetches the Description (as HTML source code) of a SimComponent from Simplace Website

## Usage

```
fetchDescriptionFromWebsite(class, version = "current")
```

## Arguments

| | |
|---|---|
| `class` | class name of the SimComponent |
| `version` | Use `current`, `trunk` or numeric version `#.#` |

## Value

string with the description text (including HTML tags)

---

`fetchSimComponentlistFromWebsite`

*Fetches the list of SimComponents from the Simplace Website*

---

## Description

Fetches the list of SimComponents from the Simplace Website

## Usage

```
fetchSimComponentlistFromWebsite(version = "current")
```

## Arguments

| | |
|---|---|
| `version` | Use `current`, `trunk` or numeric version `#.#` |

## Value

character vector with the class names of all SimComponents

## fetchSimVariablesFromWebsite *Fetches the SimVariables table for a SimComponent from Simplace Website*

### Description

Fetches the SimVariables table for a SimComponent from Simplace Website

### Usage

```
fetchSimVariablesFromWebsite(class, version = "current")
```

### Arguments

| | |
|---|---|
| class | class name of the SimComponent |
| version | Use current, trunk or numeric version #.# |

### Value

a data.frame with the SimVariables

## filterEdges *Selects edges of specific timestep*

### Description

Selects edges of specific timestep

### Usage

```
filterEdges(graph, linkage = "allsteps")
```

### Arguments

| | |
|---|---|
| graph | graph object of class dgr_graph |
| linkage | all steps, same step or previous step |

### Value

graph object of class dgr_graph

## formatProjectFile *Formats / prettyprints a project file*

### Description

Formats / prettyprints a project file

### Usage

```
formatProjectFile(
  file,
  outfile = file,
  version = "5.2",
  stripcomments = FALSE,
  dropattributes = c()
)
```

### Arguments

| | |
|---|---|
| file | filename of solution or project |
| outfile | filename the reformatted document is written to |
| version | version of the solution |
| stripcomments | if TRUE, then strip xml comments |
| dropattributes | vector of attribute names that will be dropped globally. |

### Value

no return value, called for the side effect to write a file to disk

## formatSolutionFile *Formats / prettyprints a solution file*

### Description

Formats / prettyprints a solution file

### Usage

```
formatSolutionFile(
  file,
  outfile = file,
  version = "5.2",
  stripcomments = FALSE,
  dropattributes = c()
)
```

### Arguments

| | |
|---|---|
| file | filename of solution or project |
| outfile | filename the reformatted document is written to |
| version | version of the solution |
| stripcomments | if TRUE, then strip xml comments |
| dropattributes | vector of attribute names that will be dropped globally. |

### Value

no return value, called for the side effect to write a file to disk

### Examples

```
## Not run:
original <- system.file("solution","Yield.sol.xml",package="simplaceUtil")
formatted <- tempfile(fileext = ".sol.xml")
formatSolutionFile(original, formatted, stripcomments = TRUE)

## End(Not run)
```

---

## getComponents *Gets components for a solution*

---

### Description

Gets components for a solution

### Usage

```
getComponents(x, description = TRUE)
```

### Arguments

| | |
|---|---|
| x | xml object (solution) |
| description | if TRUE, title attribute and text from description tag are included |

### Value

data.frame with the solution components (resources, sim components, outputs)

## getElementsFromSolutionFile

*Get component and links dataframe from solution file*

### Description

Get component and links dataframe from solution file

### Usage

```
getElementsFromSolutionFile(file)
```

### Arguments

| | |
|---|---|
| file | solution |

### Value

list with solution (xml2 object) and components, links and variables data.frame

## getLinkingFromComponent

*Get components that are linked from given component*

### Description

Get components that are linked from given component

### Usage

```
getLinkingFromComponent(
  graph,
  names,
  distance = 50,
  linkage = "allsteps",
  type = "all"
)
```

### Arguments

| | |
|---|---|
| graph | graph object of class dgr_graph |
| names | names (id attributes) of the selected components |
| distance | maximum distance from given components |
| linkage | all steps, same step or previous step |
| type | type of components |

### Value

graph object of class dgr_graph

---

## getLinkingToComponent *Get components that link to given components*

---

### Description

Get components that link to given components

### Usage

```
getLinkingToComponent(
  graph,
  names,
  distance = 50,
  linkage = "allsteps",
  type = "all"
)
```

### Arguments

| | |
|---|---|
| graph | graph object of class dgr_graph |
| names | names (id attributes) of selected components |
| distance | maximum distance from given components |
| linkage | all steps, same step or previous step |
| type | type of components |

### Value

graph object of class dgr_graph

---

## getLinks *Get the dataframe of links between components*

---

### Description

Get the dataframe of links between components

### Usage

```
getLinks(x, df)
```

### Arguments

| | |
|---|---|
| x | xml object (solution) |
| df | data.frame of components |

### Value

data.frame with the linked variables between components

---

## getMemoryOutputIds *Get ids of memory outputs*

---

### Description

Get ids of memory outputs

### Usage

```
getMemoryOutputIds(comp)
```

### Arguments

| | |
|---|---|
| comp | components dataframe |

### Value

character vector with the memory output ids

---

## getNeighborhood *Get the simcomponents in the neighborhood of given components*

---

### Description

Get the simcomponents in the neighborhood of given components

### Usage

```
getNeighborhood(
  graph,
  names,
  distance = 1,
  linkage = "allsteps",
  type = "all",
  set_op = "union"
)
```

### Arguments

| | |
|---|---|
| graph | graph object of class dgr_graph |
| names | names (id attributes) of the selected sim components |
| distance | maximum number of steps from given component |
| linkage | all steps, same step or previous step |
| type | type of component |
| set_op | "union" - all neighbours of selected components or "intersect" - common neighbours of all selectected components |

### Value

graph object of class dgr_graph

---

## getOutputFilenames *Get filenames of file outputs*

---

### Description

Get filenames of file outputs

### Usage

```
getOutputFilenames(comp, variables, additional = NULL)
```

### Arguments

| | |
|---|---|
| comp | components dataframe |
| variables | variables dataframe |
| additional | additional variables as named vector c("var1"="value1", ...), useful for directory placeholder |

### Value

named character vector with the filenames

## getResourceVariables *Get variables for a specific resource or transformer*

### Description

Get variables for a specific resource or transformer

### Usage

```
getResourceVariables(x, id, compactkey = FALSE)
```

### Arguments

| | |
|---|---|
| x | xml object (solution) |
| id | id of the resource or transformer |
| compactkey | if TRUE, then id and key column are merged |

### Value

a datatframe with all the variables of a resource

## getSolutionFromFile *Reads a solution from file*

### Description

Reads a solution from file

### Usage

```
getSolutionFromFile(file)
```

### Arguments

| | |
|---|---|
| file | filename of the solution |

### Value

parsed solution as xml_document

## getSolutionFromText *Reads a solution from text*

### Description

Reads a solution from text

### Usage

```
getSolutionFromText(text)
```

### Arguments

| | |
|---|---|
| text | string with xml markup |

### Value

parsed solution as xml_document

## getSolutionInfoAsDataframe *Get info for a solution*

### Description

Get info for a solution

### Usage

```
getSolutionInfoAsDataframe(file, workdir)
```

### Arguments

| | |
|---|---|
| file | solution file |
| workdir | working directory |

### Value

data.frame with all solution components as well as meta data for solution file

## getTextFromSolution *Converts a solution to text*

### Description

Converts a solution to text

### Usage

```
getTextFromSolution(sol)
```

### Arguments

| | |
|---|---|
| sol | solution object (xml_document) |

### Value

text with xml code

## getUserVariables *Gets user variables for a solution*

### Description

Gets user variables for a solution

### Usage

```
getUserVariables(x)
```

### Arguments

| | |
|---|---|
| x | xml object (solution) |

### Value

data.frame with the user defined variable

## getXMLParameter *Get a parameter value for a specific parameterset*

### Description

Get a parameter value for a specific parameterset

### Usage

```
getXMLParameter(paramdata, key, id)
```

### Arguments

| | |
|---|---|
| paramdata | parameterdata (xml2 object) |
| key | named vector c(idname="value") to identify the dataset, the name is the id of the parameter and the value has to match the parameter's value |
| id | name of the parameter of which the value should be retrieved |

### Value

the value of the parameter

### Examples

```
params <- readXMLParameterFile(system.file("input", "crop.xml", package="simplaceUtil"))
getXMLParameterIDs(params, "CROPNAME")
getXMLParameter(params, c(CROPNAME="soy bean"), "TSUM1")
getXMLParameter(params, c(CROPNAME="faba bean"), "TSUM1")
```

## getXMLParameterIDs *Get IDs of all parametersets in the parameterfile*

### Description

Get IDs of all parametersets in the parameterfile

### Usage

```
getXMLParameterIDs(paramdata, keyid)
```

### Arguments

| | |
|---|---|
| paramdata | parameterdata (xml2 object) |
| keyid | id of the parameter that identifies the parameterset |

### Value

character vector with all IDs

### Examples

```
params <- readXMLParameterFile(system.file("input", "crop.xml", package="simplaceUtil"))
getXMLParameterIDs(params, "CROPNAME")
```

---

## graphLegend *Creates legends*

---

### Description

Creates legends

### Usage

```
graphLegend(showinterfaces = TRUE, fillcolors = NULL, shapes = NULL)
```

### Arguments

| | |
|---|---|
| showinterfaces | if TRUE, include interfaces in graph |
| fillcolors | vector of colors, named by component's subtype |
| shapes | vector of shapes, named by component's type |

### Value

a list with the component legend and links (arrows) legend

---

## parseDate *Converts date (class* character*) column from output file to column of class* Date

---

### Description

Converts date (class character) column from output file to column of class Date

### Usage

```
parseDate(
  data,
  newName = "CURRENT.DATE",
  format = "%d.%m.%Y",
  oldName = "CURRENT.DATE"
)
```

### Arguments

| | |
|---|---|
| data | data.frame |
| newName | name of the transformed date column |
| format | date format |
| oldName | name of the column that holds the date to be transformed |

### Value

data.frame with date column of class Date

---

## plotLayeredOutput *Plots layered output*

---

### Description

Plots layered output

### Usage

```
plotLayeredOutput(
  data,
  column,
  simulationid = NULL,
  date_from = NULL,
  date_to = NULL,
  datecol = "CURRENT.DATE",
  nrow = NULL,
  ncol = NULL,
  sep = "_"
)
```

### Arguments

| | |
|---|---|
| data | data from memory output or file |
| column | column name used for the fill color |
| simulationid | plot only data for simulationids |
| date_from | simulation date from where values are plotted |
| date_to | simulateon date until values are plotted |
| datecol | column name for date (default CURRENT.DATE) |
| nrow | number of panel rows when plotting multiple simulation ids |
| ncol | number of panels in a row when plotting multiple simulation ids |
| sep | character that separates layer number from variable name (default "_") |

## plotScalarOutput *Plots scalar output*

### Description

Plots scalar output

### Usage

```
plotScalarOutput(
  data,
  column_x,
  columns_y,
  simulationid = NULL,
  date_from = NULL,
  date_to = NULL,
  datecol = "CURRENT.DATE",
  nrow = NULL,
  ncol = NULL
)
```

### Arguments

| | |
|---|---|
| data | data from memory output or file |
| column_x | column name for x values |
| columns_y | column name(s) for y values (vector of names) |
| simulationid | plot only data for simulationids |
| date_from | simulation date from where values are plotted |
| date_to | simulateon date until values are plotted |
| datecol | column name for date (default CURRENT.DATE) |
| nrow | number of panel rows when plotting multiple simulation ids |
| ncol | number of panels in a row when plotting multiple simulation ids |

## readXMLParameterFile *Read XML parameter file*

### Description

Read XML parameter file

### Usage

```
readXMLParameterFile(file)
```

### Arguments

| | |
|---|---|
| file | filename of parameterfile |

### Value

parameterdata (XML2 object)

---

## removeComponentInput *Removes an input for component*

---

### Description

Removes an input for component

### Usage

```
removeComponentInput(sol, componentid, id)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| componentid | id of the sim component |
| id | id of the input |

### Value

modified solution object

---

## removeNonMemoryOutputs *Removes all non-MEMORY outputs from solution.*

---

### Description

When running large amout of runs (e.g. calibration) it's recommended to avoid to write outputs on disk.

### Usage

```
removeNonMemoryOutputs(sol)
```

### Arguments

| | |
|---|---|
| sol | solution object |

### Value

modified solution object

## removeOutput *Removes output with given id*

### Description

Notice: the interface for the output will also be removed.

### Usage

```
removeOutput(sol, outputid)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| outputid | id of output to remove |

### Value

modified solution object

## removeOutputVariable *Removes an output variable from a given output*

### Description

Removes an output variable from a given output

### Usage

```
removeOutputVariable(sol, outputid, id)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| outputid | id of output from where the variable should be removed |
| id | name of the variable to remove |

### Value

modified solution object

`replaceVariable` *Replaces a variable with another.*

## Description

Notice: there is no check, whether the ids exist. Variables are replaced literarilly when they are the only content of an attribute or element content. In other attributes and element contents `${oldid}` will be replaced by `${newid}`

## Usage

```
replaceVariable(sol, oldid, newid)
```

## Arguments

| | |
|---|---|
| `sol` | solution object |
| `oldid` | id of variable to be replaced |
| `newid` | id of the replacing variable |

## Details

Notice that the id of variables from resources, SimComponents etc. should be prefixed by the enclosing resource's, SimComponent's etc. id.

## Value

modified solution object

---

`replaceVariablesWithValues`
*Replaces variables with content*

## Description

Replaces variables with content

## Usage

```
replaceVariablesWithValues(text, variables, additional = NULL)
```

## Arguments

| | |
|---|---|
| `text` | text to replace |
| `variables` | variables dataframe |
| `additional` | additional variables as named vector c("var1"="value1", ...), useful for directory placeholder |

`runSimplaceCustomGuiApp` *Runs Custom Shiny app*

## Description

User can create automatically a simple GUI for a specific solution/project by specifying solution, project, list of parameters that could be interatively changed as well a list of custom plot functions.

## Usage

```
runSimplaceCustomGuiApp(
  simplacedirs,
  solution,
  project = NULL,
  plotlist = list(Default = simplaceUtil::plotScalarOutput),
  datalist = list(Default = identity),
  paramlist = list(),
  paramtransform = identity,
  ...
)
```

## Arguments

| | |
|---|---|
| `simplacedirs` | a list containing the named elements instdir, workdir, outdir and options |
| `solution` | filename of a solution (or solution as text) |
| `project` | filename of a project |
| `plotlist` | list of named functions that produce a plot |
| `datalist` | list of named functions that transform output dataframes |
| `paramlist` | list of named parameter elements that will be used for input sliders / option lists |
| `paramtransform` | function to transform the parameterlist |
| `...` | parameters passed to shiny::runApp |

## Details

- `plotlist` takes a named list of functions of the form `f(data, xvalue, yvalues, simulationids, from, to)` and creates a plot
  - `data` is a data.frame
  - `xvalue` is a string with the column name for x
  - `yvalues` is a vector of strings with the column names for y
  - `simulationids` can be NULL or a vector of strings with simulation ids
  - `from` and `to` are dates for subsetting the data that will be plotted
- `datalist` takes a named list of functions of the form `f(data)` and return the modified data.frame

- `paramlist` is a named list of parameters where the name of each element has to match a `var` in the solution (unless the list will be transformed). Each parameter is a list with the elements
  - `label`
  - `value`
  - `min` and `max` or `options` (vector of possible values)
- `paramtransform` is a function that transforms the input parameters to hand them over to Simplace

## Value

does not return a value, called for the side effect of running the shiny app

## Examples

```
## Not run:
instdir <- simplace::findFirstSimplaceInstallation()
simplacedirs <- list(
  instdir = instdir,
  workdir = paste0(instdir,"/simplace_run/simulation/"),
  outdir = paste0(instdir,"/simplace_run/output/")
)

solution <- paste0(simplacedirs$workdir,"/gk/solution/calibration/Yield.sol.xml")

paramlist <- list(
 vLUE = list(
   label = "Light Use efficiency",
   value = 3.0,
   min = 1.0,
   max = 6.5
 )
)

plotlist <- list(
  "Default" = simplaceUtil::plotScalarOutput,
  "Boxplots" = function(data,x,y,sim,from,to) {
    if(length(y)==1 && is.numeric(data[[y]])) {
      boxplot(formula(paste(y,"~simulationid")),data)
    }
  }
)

datalist <- list(
  "Aggregated Yield"=function(data) {
    dplyr::group_by(data, simulationid) |>
    dplyr::summarise(minY=min(Yield),maxY=max(Yield))},
  "Raw Data"=identity
)

simplaceUtil::runSimplaceCustomGuiApp(
  simplacedirs, sol,
```

```
  plotlist = plotlist,
  datalist = datalist,
  paramlist = paramlist
)


## End(Not run)
```

---

`runSimplaceGuiApp` *Runs Shiny app*

---

## Description

Runs Shiny app

## Usage

```
runSimplaceGuiApp(...)
```

## Arguments

| | |
|---|---|
| `...` | parameters passed to shiny::runApp |

---

`scaleXMLParameter` *Scales a parameter value by a value*

---

## Description

Scales a parameter value by a value

## Usage

```
scaleXMLParameter(paramdata, key, id, factor = 1)
```

## Arguments

| | |
|---|---|
| `paramdata` | parameterdata (xml2 object) |
| `key` | named vector `c(idname="value")` to identify the dataset, the name is the id of the parameter and the value has to match the parameter's value |
| `id` | name of the parameter which should be scaled |
| `factor` | scaling factor |

## Value

modified parameter set (xml2 object)

## Examples

```
params <- readXMLParameterFile(system.file("input", "crop.xml", package="simplaceUtil"))
getXMLParameter(params, c(CROPNAME="soy bean"), "RUETableRUE")
params_new <-scaleXMLParameter(params, c(CROPNAME="soy bean"), "RUETableRUE",0.9)
getXMLParameter(params_new, c(CROPNAME="soy bean"), "RUETableRUE")
```

---

`setInputValue` *Sets the value of an input for a given parent element*

---

## Description

Sets the value of an input for a given parent element

## Usage

```
setInputValue(sol, parentid, id, value, datatype = NULL)
```

## Arguments

| | |
|---|---|
| `sol` | solution object |
| `parentid` | id of input parent |
| `id` | id of the input |
| `value` | new value |
| `datatype` | datatype (optional) |

## Value

modified solution object

---

`setInputValueForCategory`
*Sets the value of an input in all elements of the given category*

---

## Description

It's main use is to change consistently inputs of all transformers, e.g. `layerthickness`. It can also be used to set sim component inputs to a fixed value.

## Usage

```
setInputValueForCategory(sol, category, id, value, datatype = NULL)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| category | tag of categories containing inputs, e.g. transform or simcomponent |
| id | id of the input |
| value | new value |
| datatype | datatype (optional) |

### Value

modified solution object

---

## setXMLParameter *Set a parameter value for a specific parameterset*

---

### Description

Set a parameter value for a specific parameterset

### Usage

```
setXMLParameter(paramdata, key, id, value = "")
```

### Arguments

| | |
|---|---|
| paramdata | parameterdata (xml2 object) |
| key | named vector c(idname="value") to identify the dataset, the name is the id of the parameter and the value has to match the parameter's value |
| id | name of the parameter of which the value should be set |
| value | new value to set (vector) |

### Value

modified parameter set (xml2 object)

### Examples

```
params <- readXMLParameterFile(system.file("input", "crop.xml", package="simplaceUtil"))
getXMLParameter(params, c(CROPNAME="soy bean"), "TSUM1")
params_new <-setXMLParameter(params, c(CROPNAME="soy bean"), "TSUM1",400)
getXMLParameter(params_new, c(CROPNAME="soy bean"), "TSUM1")
```

## simplaceUtil *Package with utility function for working with Simplace*

### Description

Provides functions to work with the modeling framework Simplace

### Details

- get elements of a simplace solution as dataframes
- visualise the structure of a solution as a graph
- helper functions for transforming output data
- simplified plot functions for simulation output variables
- shinyApp to visualise graphs, run simulations and plot results

### Author(s)

Gunther Krauss

### Examples

```
## Not run:
# run the GUI
runSimplaceGuiApp()

## End(Not run)

# visualise the solution structure as graph
graph <- solutionToGraph(system.file("solution", "Yield.sol.xml",
  package = "simplaceUtil"))
DiagrammeR::render_graph(graph)
```

## solutionToGraph *Creates a graph from a solution*

### Description

Creates a graph from a solution

### Usage

```
solutionToGraph(
  file,
  showinterfaces = FALSE,
  fillcolors = NULL,
  shapes = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| file | solution file |
| showinterfaces | if true include interfaces |
| fillcolors | vector of colors, named by component's subtype |
| shapes | vector of shapes, named by component's type |
| ... | options passed to DiagrammeR::create_graph() |

### Value

graph object of class dgr_graph

---

## swapComponents *Swaps the order of SimComponents in the solution*

---

### Description

Rearranges the order of SimComponents. All components that are mentioned in the vector order will be rearranged according to their position in the order vector. All other components remain on the same position. E. g. an order of c(5,6,1,3) will put components on position 1,3,5,6 to position 5, 6, 1, 3.

### Usage

```
swapComponents(sol, order)
```

### Arguments

| | |
|---|---|
| sol | solution object |
| order | a vector of component positions |

### Value

modified solution object

## transformLayeredData *Transform layered output data in long format*

### Description

Transform layered output data in long format

### Usage

```
transformLayeredData(data, sep = "_")
```

### Arguments

| | |
|---|---|
| data | dataframe |
| sep | character that separates layer number from variable name (default "_") |

### Value

dataframe in long format

## unitsymbols *Symbols for unit URIs*

### Description

Symbols for unit URIs

### Usage

```
data(unitsymbols)
```

### Format

A named vector of unit symbols where the names ar the unit URIs

### Examples

```
# data(unitsymbols)
# unitsymbols['http://www.wurvoc.org/vocabularies/om-1.8/gram_per_square_metre']
```

## writeSolutionToFile *Writes solution to file*

### Description

Writes solution to file

### Usage

```
writeSolutionToFile(sol, file)
```

### Arguments

| | |
|---|---|
| sol | solution object (xml_document) |
| file | filename for the solution |

### Value

nothing, function writes a file as a side effect

## writeXMLParameterFile *Write parameterdata to xml file*

### Description

Write parameterdata to xml file

### Usage

```
writeXMLParameterFile(paramdata, file)
```

### Arguments

| | |
|---|---|
| paramdata | parameterdata (xml2 object) |
| file | filename of parameterfile |

### Value

just for side effects, returs paramdata invisibly

# Index


∗ **datasets**
    unitsymbols, 42

addComponentInput, 3
addCSVOutput, 4
addMemoryOutput, 5
addOutputVariable, 5
addTimingSimComponent, 6
addUserVariable, 7
addXMLParameter, 8

changeAllOutputTypesToMemory, 8
changeInterfaceFile, 9
changeOutputType, 10
componentsToGraph, 10
createCodeStubsForSimVariables, 11
createResourceStubsFromCsv, 12
createResourceStubsFromXml, 13

DiagrammeR::create_graph(), *11*

enhanceFunctionWithBoundaries, 14
enhanceFunctionWithPenalty, 15

fetchDescriptionFromWebsite, 17
fetchSimComponentlistFromWebsite, 17
fetchSimVariablesFromWebsite, 18
filterEdges, 18
formatProjectFile, 19
formatSolutionFile, 19

getComponents, 20
getElementsFromSolutionFile, 21
getLinkingFromComponent, 21
getLinkingToComponent, 22
getLinks, 22
getMemoryOutputIds, 23
getNeighborhood, 23
getOutputFilenames, 24
getResourceVariables, 25
getSolutionFromFile, 25
getSolutionFromText, 26
getSolutionInfoAsDataframe, 26
getTextFromSolution, 27
getUserVariables, 27
getXMLParameter, 28
getXMLParameterIDs, 28
graphLegend, 29

parseDate, 29
plotLayeredOutput, 30
plotScalarOutput, 31

readXMLParameterFile, 31
removeComponentInput, 32
removeNonMemoryOutputs, 32
removeOutput, 33
removeOutputVariable, 33
replaceVariable, 34
replaceVariablesWithValues, 34
runSimplaceCustomGuiApp, 35
runSimplaceGuiApp, 37

scaleXMLParameter, 37
setInputValue, 38
setInputValueForCategory, 38
setXMLParameter, 39
simplaceUtil, 40
simplaceUtil-package *(*simplaceUtil*)*, 40
solutionToGraph, 40
swapComponents, 41

transformLayeredData, 42

unitsymbols, 42

writeSolutionToFile, 43
writeXMLParameterFile, 43